On Using UML Profiles in ATL Transformations*

Manuel Wimmer and Martina Seidl

Business Informatics Group,
Vienna University of Technology, Austria
{wimmer|seidl}@big.tuwien.ac.at

Abstract. For defining modeling languages, metamodels and UML profiles are the proposed options. While metamodels are supported by several dedicated model transformation approaches, currently no transformation language exists which supports UML profiles as first class language definitions. Instead, the usage of UML profiles in transformations is implicit by using calls to external UML APIs.
In this paper, we first discuss the state-of-the-art of using UML profiles in ATL. Subsequently, three approaches for supporting profiles as first class language definitions within ATL transformations are introduced and discussed. In particular, these approaches aim at using stereotypes and tagged values within declarative rules without using external API calls. The benefits are: first, the enhanced static checking of ATL transformations, second, the more explicit representation of transformations logic enhances the application of higher-order transformations, and third, enhanced tool support such as code completion may be provided.

1 Introduction

Context. For defining languages in the field of model engineering, metamodels and UML profiles are the proposed options. While metamodels, mostly based on the Meta Object Facility (MOF) [3], allow the definition of languages from scratch, UML profiles are used to extend UML [4] with new concepts.

Problem. Metamodels are supported by current model transformation languages as first class language definitions, however, UML profiles are not. Nevertheless, UML profiles may be used in transformations with a little work-around. UML profiles are considered as additional input models and by calling external UML APIs, profiles are applied. Although this is a technical possibility, the development of such transformations code is challenging (cf. Section 4).

Solution. As a first step of using UML profiles as first class language definitions in model transformations, we discuss the state-of-the-art of using UML profiles in ATL [1], identify some shortcomings, and propose three ways of using UML profiles in ATL transformations more systematically. We demonstrate this

* This work has been funded by the Austrian Science Fund (FWF) under grant P21374-N13.

by employing a running example. The explicit use of profiles as language definitions provides various benefits. First, errors concerning the misuse of profiles may be detected at design time. Second, transformations may be easier enhanced by higher-order transformations. Third, tooling issues may be improved, e.g., code completion for recommending applicable stereotypes.

2 Motivating Example

Two main scenarios exist where UML models annotated with profile information have to be transformed. The first one is the vertical transformation scenario by following the model-driven architecture proposed by the OMG. Platform independent models are created which are subsequently refined with platform specific information by applying profiles consisting of stereotypes and tagged values for specific platforms. From these platform specific models, code is generated where the profile information is one of the main driver for the code generation. The second scenario is the horizontal transformation scenario, where modeling languages have to be bridged to UML. For example, in the context of the ModelCVS project [2], one industry partner was using the CASE tool AllFusion Gen[1] (AFG) from ComputerAssociate which supports a language for designing data-intensive applications and provides sophisticated code generation facilities. Due to modernization of the IT department and the search for an exit strategy (if tool support is no longer guaranteed), the need arises to extract models from the legacy tool and import them into UML tools while at the same time the code generation of AllFusion Gen should in the future be usable for UML models as well. Therefore, models have to be exchanged between AFG and UML tools without loss of information which requires the extension of UML.

Running Example. As a running example, we are using an excerpt of a tool integration case study conducted in the ModelCVS project. The goal was to bridge the structural modeling part of AFG and UML, i.e., the AFG Data Model with the UML Class Diagram. Since AllFusion Gen's data model is based on the ER model, it supports ER modeling concepts like *EntityTypes*, *Attributes*, and *Relationships*. Furthermore, two concrete subtypes of the abstract *EntityType* concept can be distinguished, namely *AnalysisEntityType* and *DesignEntityType*. AllFusion Gen is typically used for modeling data intensive applications which make excessive use of database technologies. Therefore, the data model allows the definition of platform specific information typically usable for generating optimized database code, e.g., *EntityTypes* have special occurrence configurations. It is obvious that the corresponding UML model type for AllFusion Gen's data model is the class diagram.

3 State-of-the-Art in ATL

After having introduced the modeling languages to be integrated, we now proceed with bridging AFG with UML. Due to space limitations, we only consider

[1] http://ca.com/us/products/product.aspx?ID=256 (accessed 6 June 2009)

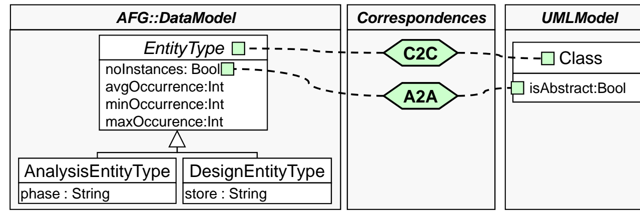

Fig. 1. EntityType to Class mapping

an excerpt which is used in the subsequent ATL listings. Details for the *EntityType_2_Class* mapping are illustrated in Fig. 1. The abstract metaclass *EntityType* of AFG is mapped to the metaclass *Class* in UML. In addition, the attribute *noInstances* on the LHS is mapped to the attribute *isAbstract* on the RHS. Several platform specific attributes of AFG remain unmapped which have to be represented as tagged values in the UML profile.

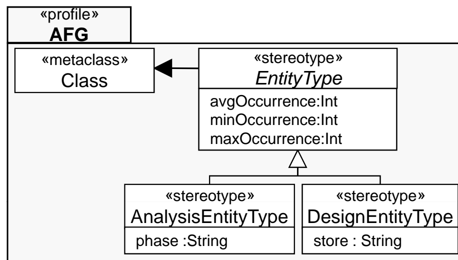

Fig. 2. AFG profile excerpt

AFG Profile. The profile excerpt resulting from the *EntityType_2_Class* mapping is shown in Fig. 2. An abstract stereotype *EntityType* is defined for the abstract metaclass *EntityType*. Furthermore, unmapped attributes are represented as tagged values. Two concrete sub stereotypes are defined in addition which have tagged values attached for their unmapped attributes.

Transformation Architecture. Currently the only way to make use of UML profiles in model transformations is that the profiles are additional input models for the model transformation. Fig. 3 illustrates this by showing the runtime configuration for transforming AFG models into UML with ATL. The user has to define the AFG model as first input model, and the UML profile as second input model. The output model is the UML model, which both conforms to the UML metamodel and to the UML profile, which also conforms to the UML metamodel. This configuration reveals the problem, that UML profiles are located on the *M1* layer according to the OMG metamodeling layers (because they conform to the UML metamodel being located at the M2 layer), and also on the M2 layer (because UML models are located at the M1 layer and instantiate the profile, which is thereby located at the M2 layer).

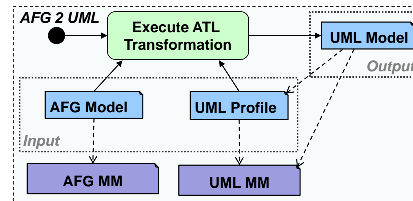

Fig. 3. ATL runtime configuration

ATL Transformation Code. Listing 1.1 illustrates an excerpt of the transformation from AFG to UML. It consists of an abstract transformation rule, which results from the *EntityType_2_Class* mapping. The current ATL version does not allow to define *do* blocks for super rules, thus, feature to tagged value mappings, e.g., for *avgOccurrence*, must be defined in concrete sub rules. In fact, two concrete subrules are necessary for our running example, one for transforming *AnalysisEntityTypes* (cf. second rule in Listing 1.1) and one for *DesignEntityTypes* (not shown due to space limitations). These subrules have to implement stereotype applications and feature to tagged value assignments for super stereotype tagged values (e.g., *avgOccurrence*) as well as for leaf stereotype tagged values (e.g., *phase*).

Listing 1.1. AFG to UML excerpt

```
1  module AFG2UML;
2  create OUT:UML from IN:AFG, IN2:PRO;
3
4  helper def: ste : PRO!Stereotype = PRO!Stereotype.allInstances()
5    -> select(e|e.name = 'AnalysisEntityType').first();
6
7  abstract rule ET_2_Class {
8    from s : AFG!EntityType
9    to t : UML!Class ( isAbstract <- s.noInstances )
10 }
11 rule AnalysisET_2_Class extends ET_2_Class{
12   from s : AFG!AnalysisEntityType
13   to t : UML!Class
14   do{
15    t.applyStereotype(ste);
16    --assign tagged values from super stereotypes
17    if(not s.avgOccurrence.oclIsUndefined()){
18      t.setTaggedValue(ste,'avgOccurrence',s.avgOccurrence);
19    }
20    ...
21    --assign tagged values from leaf stereotype
22    ...
23   }
24 }
```

4 Profile-Aware Transformation Language

In this section we first discuss some shortcomings of ATL concerning the usage of UML profiles based on the afore presented listing and present three approaches for tackling these shortcomings.

Shortcomings. When taking a closer look on the afore shown ATL listing, the following shortcomings may be identified. 1) Feature to tagged value assignments have to be done in *do* blocks. Because *do* blocks cannot be used for super rules, these assignments have to be done for each sub rule again and again. 2) The application of stereotypes is only implicit by calling external UML APIs. Therefore, it cannot be checked if a certain stereotype is applicable for the UML element and because of stereotypes are only encoded as a String values, it is not ensured that the stereotype actually exists in the profile. 3) The same problems as for stereotype assignments exist for tagged value assignments. 4) Some low-level details of external UML APIs have to be considered in the transformations.

For example, the assignment of a null value to a tagged value results in a runtime exception.

Profile-aware Transformation Language. Now, three approaches for using UML profiles as first class language definitions are proposed and discussed.

(1) Merge. This approach is a lightweight approach, meaning that no ATL language modification is necessary. Instead of using the UML metamodel and the UML profile as separated definitions, they are merged into one metamodel. For this, stereotypes become metaclasses, tagged values become features, and extension relationships become inheritance relationships. In Listing 1.2, the ATL code is shown which is capable of transforming AFG models into UML models which conform to the merged metamodel. Please note that the transformation code is more concise compared to Listing 1.1. Advantages of this approach are as follows. The merged metamodel is automatically created and it is not necessary to extend the ATL language with new syntax elements. However, there are also some drawbacks. In addition to the merged metamodel, model adapters are needed to transform UML models conforming to the merged metamodel into standard UML models using profiles. Furthermore, in cases where more than one stereotype is applicable on the same element, this approach is not sufficient.

Listing 1.2. AFG to UML (merged UML metamodel)

```
1  module AFG2UML;
2  create OUT:UML_ext from IN:AFG;
3
4  abstract rule ET_2_ET {
5    from s : AFG!EntityType
6    to t : UML!EntityType (
7      isAbstract <- s.noInstances,
8      avgOccurrence <- s.avgOccurrence,
9      ...
10   )
11 }
12 rule AnalysisET_2_AnalysisET extends ET_2_ET{
13   from s : AFG!AnalysisEntityType
14   to t : UML!AnalysisEntityType( phase <- s.phase )
15 }
```

(2) Preprocessor. This approach uses a slightly modified ATL syntax for describing UML profile aware transformations and a preprocessor which creates standard ATL transformations for execution purposes. The modified ATL syntax is used in Listing 1.3. Please note that we have introduced a *using* keyword in the header definition for referencing the used profile and an *apply* keyword which is used in the *to* part for applying stereotypes on target elements. For this modified ATL syntax, we are now able to provide dedicated code completion and static validation to ensure the proper usage of UML profiles. Furthermore, we allow to use feature to tagged value assignments in the *to* parts of the transformations, thus the inheritance feature between declarative rules can be fully exploited. Advantages of this approach are that only a modified syntax of ATL has to be provided as well as a transformation to standard ATL. Disadvantages are that debugging is only supported for the generated standard ATL transformations and a new development line is created that requires a parallel development with the standard ATL development line.

(3) Extending ATL. Finally, the additional syntax elements for using UML profiles can be directly integrated in ATL. This requires to extend not only the syntax, but also the ATL compiler which is much more implementation work compared to the previous approaches. However, the benefit is that neither a preprocessing of metamodels and models (first approach) nor of ATL code (second approach) is necessary. In addition, the complete tool support of ATL, e.g., the debugger, may be used.

Listing 1.3. AFG to UML (extended ATL syntax)

```
1  module AFG2UML;
2  create OUT:UML using AFG_Profile from IN:AFG;
3
4  abstract rule ET_2_ET {
5    from s : AFG!EntityType
6    to t : UML!Class apply EntityType (
7      isAbstract <- s.noInstances,
8      avgOccurrence <- s.avgOccurrence,
9      ...
10   )
11 }
12 rule AnalysisET_2_AnalysisET extends ET_2_ET{
13   from s : AFG!AnalysisEntityType
14   to t : UML!Class apply AnalysisEntityType(
15     phase <- s.phase
16   )
17 }
```

5 Conclusions and Future Work

In this paper, we discussed the state-of-the-art of using UML profiles within ATL transformations, identified some shortcomings, and proposed three approaches how to use UML profiles as first class language definitions. In future work, we plan to realize experimental ATL versions for UML profiles by following the three presented approaches. Furthermore, we plan to evaluate vertical ATL transformations in which UML profiles are heavily used.

References

1. F. Jouault and I. Kurtev. Transforming Models with ATL. In *Proceedings of Satellite Events at the MoDELS 2005 Conference, Jamaica*, 2006.
2. E. Kapsammer, H. Kargl, G. Kramler, G. Kappel, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer. On Models and Ontologies - A Semantic Infrastructure Supporting Model Integration. In *Proceedings of Modellierung 2006, Austria*, 2006.
3. Object Management Group. Meta Object Facility (MOF) 2.0 Core Specification Version 2.0. http://www.omg.org/docs/formal/06-01-01.pdf, Oct. 2004.
4. Object Management Group. UML Specification: Superstructure Version 2.0. http://www.omg.org/docs/formal/05-07-04.pdf, August 2005.